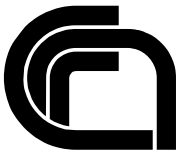


Consiglio Nazionale delle Ricerche

Superfast solution of Toeplitz-like systems

P. Favati G. Lotti O. Menchi

IIT TR-24/2011

Technical report

Ottobre 2011

Istituto di Informatica e Telematica

Superfast solution of Toeplitz-like systems

P. Favati G. Lotti O. Menchi

Abstract

In this paper a new $O(N \log^3 N)$ solver for $N \times N$ Toeplitz-like systems, based on a divide and conquer technique, is presented. Similarly to the superfast algorithm MBA for the inversion of a Toeplitz-like matrix [2, 16], it exploits the displacement properties. In order to avoid the well known numerical instability of the explicit inversion, the new algorithm relies on the triangular factorization and back-substitution formula for the system seen as a 2×2 block system with blocks of half size. The same idea has been used in [19] to improve the numerical stability of superfast methods based on the generalized Schur algorithm for positive definite Toeplitz matrices, but the algorithm we propose can be applied also to nonsymmetric Toeplitz-like systems. The stability of the algorithm is examined through numerical experiments.

1 Introduction

Toeplitz systems arise frequently in linear algebra (see [4] for a list of possible sources) and special low cost algorithms have been devised to solve them. Unfortunately, when a simple operation like multiplication or inversion or low rank modification is applied to a Toeplitz matrix, the Toeplitz structure is lost and more general structures must be considered. The class of Toeplitz-like matrices, which is closed for the most common operations applied in numerical algorithms, seems more suitable from this point of view. In this paper the problem of solving the linear system

$$A\boldsymbol{x} = \boldsymbol{b}, \tag{1}$$

where A is an $N \times N$ nonsingular Toeplitz-like matrix and $\boldsymbol{b}$ is a vector of size N, is addressed.

For the solution of Toeplitz-like systems, superfast algorithms (i.e. algorithms having a complexity of order less than N^2) have been proposed. The ones described in [2, 16] are based on the concept of displacement rank (see [11, 13, 14, 15]) and have $O(N \log^2 N)$ complexity. Other superfast algorithms based on the generalized Schur algorithm have been developed (see [1, 3, 7, 17]), but their use is limited to the symmetric case. All these algorithms for the solution of Toeplitz-like systems compute the generators of the inverse of the coefficient matrix and then multiply the inverse by the right-hand side vector, using the FFT. It is well known that methods based on the explicit inversion

of the coefficient matrix are usually unstable (see [12]). More stable methods compute the solution directly, avoiding the computation of the inverse matrix and the subsequent matrix-vector multiplication. For example, the widely used Levinson algorithm, whose stability is analyzed in [4, 6, 8], belongs to this class of methods, but is a fast algorithm (i.e. with $O(N^2)$ complexity).

We are interested in devising a more stable superfast algorithm for problem (1), based on the concept of displacement rank. Following the same idea of [19], we propose a method which solves the system by combining the solutions of two half size Toeplitz-like systems with enlarged right-hand side, according to a divide and conquer strategy. As for the method of [19], the algorithm proposed here avoids the explicit inversion of the matrix, but at the cost of increasing the complexity from $O(N \log^2 N)$ to $O(N \log^3 N)$ floating point operations. The increasing in complexity occurs because the right-hand sides are increased by a constant number (proportional to the displacement rank of the given matrix) of columns at each recursion step. In contrast to the stabilized algorithm of [19], which works on definite positive Toeplitz systems with $O(N \log N)$ storage requirement, our algorithm can be applied to general Toeplitz-like systems and requires only $O(N)$ storage.

To fully exploit the displacement structure of matrix A, we need to consider instead of problem (1) the more general problem of simultaneously solving

$$A\boldsymbol{x} = \boldsymbol{b}, \quad A^T\boldsymbol{y} = \boldsymbol{b}_t,$$

where $\boldsymbol{b}_t$ is a vector of size N. In Section 2, for the sake of clarity, a first version of the proposed algorithm which does not yet make use of the displacement structure is described. The properties of Toeplitz-like matrices, together with the explicit expressions for the generators of the matrices used by the algorithm are given in Section 3. The description of the final version of the algorithm, fully exploiting the displacement properties of the problem, is given in Section 4. Its computational complexity and storage requirement are discussed in Section 5. The numerical experiments of Section 6 show that the proposed algorithm outperforms the superfast algorithm of [2, 16] from the stability point of view and that it does not lose too much precision compared with the fast Levinson algorithm [9, 13]. Moreover, it appears that the algorithm is reliable when the conditioning of the involved matrices is not too large.

2 The basic recursive algorithm

System (1) can be solved recursively by exploiting the partition of A into square submatrices of order $N/2$ and the compatible partition of vectors $\boldsymbol{x}$ and $\boldsymbol{b}$

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \quad \boldsymbol{x} = \begin{bmatrix} \overline{\boldsymbol{x}} \\ \underline{\boldsymbol{x}} \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} \overline{\boldsymbol{b}} \\ \underline{\boldsymbol{b}} \end{bmatrix}. \tag{2}$$

In fact the solution can be found by computing

$$\begin{aligned}
&\left[\boldsymbol{u}\,, F\right] = A_{11}^{-1}\left[\overline{\boldsymbol{b}}\,, A_{12}\right],\\
&\boldsymbol{y} = \underline{\boldsymbol{b}} - A_{21}\boldsymbol{u}, \quad S = A_{22} - A_{21}F,\\
&\underline{\boldsymbol{x}} = S^{-1}\boldsymbol{y}, \quad \overline{\boldsymbol{x}} = \boldsymbol{u} - F\underline{\boldsymbol{x}}.
\end{aligned} \tag{3}$$

Hence the computation of the solution of an $N \times N$ system can be performed by solving two $N/2 \times N/2$ subsystems, the first one with matrix A_{11} and an increased number of right-hand sides, the second one with the Schur complement S of A_{11} without increasing the number of right-hand sides.

We assume $N = 2^p\, n_e$, where n_e represents the size of the systems to be solved at the last level of recursion. For the sake of simplicity, we restrict our analysis to the case $n_e = 2^q$, with $q < p$, but our results hold also in the general case.

As already said, we need to solve at the same time also the system with the transposed matrix. The procedure (3) can be generalized to solve systems of any number of right-hand sides of the form

$$A^{(k)}X^{(k)} = B^{(k)}, \quad A^{(k)T}X_t^{(k)} = B_t^{(k)} \tag{4}$$

where k denotes the recursion level, $A^{(k)}$ is a square matrix of size 2^{p-k} and $B^{(k)}$, $B_t^{(k)}$ are the right-hand sides having together ν columns (initially $k = 0$ and $\nu = 1$). Setting

$$A^{(k)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ A_{21}^{(k)} & A_{22}^{(k)} \end{bmatrix}, \quad X^{(k)} = \begin{bmatrix} \overline{X}^{\,(k)} \\ \underline{X}^{(k)} \end{bmatrix}, \quad B^{(k)} = \begin{bmatrix} \overline{B}^{\,(k)} \\ \underline{B}^{(k)} \end{bmatrix}, \tag{5}$$

and

$$X_t^{(k)} = \begin{bmatrix} \overline{X}_t^{\,(k)} \\ \underline{X}_t^{\,(k)} \end{bmatrix}, \quad B_t^{(k)} = \begin{bmatrix} \overline{B}_t^{\,(k)} \\ \underline{B}_t^{\,(k)} \end{bmatrix}.$$

from (3) we have

$$\begin{aligned}
&\left[U\,, F\right] = A_{11}^{(k)-1}\left[\overline{B}^{\,(k)}\,, A_{12}^{(k)}\right], \quad \left[U_t\,, G^T\right] = A_{11}^{(k)-T}\left[\overline{B}_t^{\,(k)}\,, A_{21}^{(k)T}\right],\\
&Y = \underline{B}^{(k)} - A_{21}^{(k)}U, \quad Y_t = \underline{B}_t^{\,(k)} - A_{12}^{(k)T}U_t, \quad S = A_{22}^{(k)} - A_{21}^{(k)}F,\\
&\underline{X}^{(k)} = S^{-1}Y, \quad \underline{X}_t^{\,(k)} = S^{-T}Y_t,\\
&\overline{X}^{\,(k)} = U - F\underline{X}^{(k)}, \quad \overline{X}_t^{\,(k)} = U_t - G^T\underline{X}_t^{\,(k)}.
\end{aligned}$$

The following recursive routine `solvemat` shows how the computation can be performed. The first time, the function is applied with $k = 0$, matrix $A^{(0)} = A$, right-hand sides $B^{(0)} = \boldsymbol{b}$ and $B_t^{(0)}$ a void vector. Subsequent calls are made to matrices of halved dimension, until the preassigned size $n_e = 2^q$ is reached. At the kth level of recursion, with $k = 0, \ldots, p - q$, there are 2^k pairs

of systems of the form (4) with a matrix of size 2^{p-k}, which we assume to be nonsingular. At the last step there are 2^{p-q} pairs of systems with matrices of size 2^q. These last step matrices are called elementary and the corresponding systems are solved directly, by means of a routine `elem` (implementing, for example, Gauss method). The partition of the matrix $A^{(k)}$ and of the right-hand side $B^{(k)}$ is performed according to (5) using the colon notation.

$\texttt{function}\ \left[X^{(k)}, X_t^{(k)}\right] = \texttt{solvemat}\left(k, A^{(k)}, B^{(k)}, B_t^{(k)}\right)$

$\texttt{if}\ \ k = p - q$

$[\,X^{(k)}, X_t^{(k)}\,] = \ \texttt{elem}\,(A^{(k)}, B^{(k)}, B_t^{(k)});$

$\texttt{else}$

$m = 2^{p-k}; \quad r_1 = (1 : m/2); \quad r_2 = (m/2 + 1 : n);$

$A_{11} = A^{(k)}(r_1, r_1); \quad A_{12} = A^{(k)}(r_1, r_2);$

$A_{21} = A^{(k)}(r_2, r_1); \quad A_{22} = A^{(k)}(r_2, r_2);$

$\overline{B} = B^{(k)}(r_1, :); \quad \overline{B}_t = B_t^{(k)}(r_1, :);$

$\underline{B} = B^{(k)}(r_2, :); \quad \underline{B}_t = B_t^{(k)}(r_2, :);$

$B = \left[\,\overline{B}\,, A_{12}\right]; \quad B_t = \left[\,\overline{B}_t\,, A_{21}^T\right];$

$\left[W\,, W_t\right] = \texttt{solvemat}\,(k+1, A_{11}, B,\ B_t);$

$[\,U, F\,] = W; \quad [\,U_t, G_t\,] = W_t;$

$Y = \underline{B} - A_{21}U; \quad Y_t = \underline{B}_t - A_{12}U_t; \quad S = A_{22} - A_{21}F;$

$\left[\,\underline{X}\,,\, \underline{X}_t\right] = \texttt{solvemat}\,(k+1, S\,, Y\,, Y_t);$

$\overline{X} = U - F\underline{X}; \quad \overline{X}_t = U_t - G_t\underline{X}_t;$

$X^{(k)} = \begin{bmatrix} \overline{X} \\ \underline{X} \end{bmatrix}; \quad X_t^{(k)} = \begin{bmatrix} \overline{X}_t \\ \underline{X}_t \end{bmatrix};$

$\texttt{end}$

The divide and conquer strategy used for the function `solvemat` leads to a factorization of the matrix $A = A^{(0)}$. In fact, from (5) we have

$$A^{(0)} = \begin{bmatrix} A_{11}^{(0)} & A_{12}^{(0)} \\ A_{21}^{(0)} & A_{22}^{(0)} \end{bmatrix} = \begin{bmatrix} A_{11}^{(0)} & O \\ A_{21}^{(0)} & S \end{bmatrix} \begin{bmatrix} I & F \\ O & I \end{bmatrix},$$

where $F = A_{11}^{(0)-1}A_{12}^{(0)}$ and $S = A_{22}^{(0)} - A_{21}^{(0)}F$. Denoting $\mathcal{L}_1^{(0)} = A^{(0)}$, $\mathcal{L}_1^{(1)} = A_{11}^{(0)}$, $\mathcal{L}_2^{(1)} = S$, $\mathcal{U}_1^{(1)} = F$ and $\divideontimes$ a suitable matrix of consistent size, we write

$$\mathcal{L}_1^{(0)} = \begin{bmatrix} \mathcal{L}_1^{(1)} & O \\ \divideontimes & \mathcal{L}_2^{(1)} \end{bmatrix} \begin{bmatrix} I & \mathcal{U}_1^{(1)} \\ O & I \end{bmatrix}.$$

Analogous relations hold for $\mathcal{L}_1^{(1)}$ and $\mathcal{L}_2^{(1)}$, hence

$$\mathcal{L}_1^{(0)} = \left[\begin{array}{cc|cc} \mathcal{L}_1^{(2)} & O & \multicolumn{2}{c}{\multirow{2}{*}{O}} \\ \divideontimes & \mathcal{L}_2^{(2)} & & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{$\divideontimes$}} & \mathcal{L}_3^{(2)} & O \\ \multicolumn{2}{c|}{} & \divideontimes & \mathcal{L}_4^{(2)} \end{array}\right] \left[\begin{array}{cc|cc} I & \mathcal{U}_1^{(2)} & \multicolumn{2}{c}{\multirow{2}{*}{$\divideontimes$}} \\ O & I & & \\ \hline \multicolumn{2}{c|}{\multirow{2}{*}{O}} & I & \mathcal{U}_3^{(2)} \\ \multicolumn{2}{c|}{} & O & I \end{array}\right].$$

In general, setting

$$\mathcal{L}_j^{(k)} = \begin{bmatrix} \mathcal{L}_{2j-1}^{(k+1)} & O \\ \divideontimes & \mathcal{L}_{2j}^{(k+1)} \end{bmatrix} \begin{bmatrix} I & \mathcal{U}_{2j-1}^{(k+1)} \\ O & I \end{bmatrix},$$

we see that after $p-q$ recursive steps $A^{(0)} = \mathcal{L}\mathcal{U}$, where $\mathcal{L}$ is a block lower triangular matrix, with the blocks $\mathcal{L}_1^{(p-q)}, \ldots, \mathcal{L}_{2^{p-q}}^{(p-q)}$ of size 2^q on the diagonal and $\mathcal{U}$ is a block upper triangular matrix, with 2^{p-q} identity blocks of size 2^q on the diagonal. This factorization is the same we would obtain by applying to A a block Gaussian elimination by columns without pivoting.

Of course this result holds only in a pure theoretical context. When the algorithm is applied in practice, i.e. in a finite arithmetical environment where associative and distributive properties do not hold for the multiplication of matrices, the factorization may be different from the one given by the Gaussian elimination, but we believe that the stability analysis usually conducted for the Gaussian elimination [12] can be applied with minor modifications in the case of the function `solvemat`. The equivalence, from the stability point of view, of `solvemat` and block Gaussian elimination by columns without pivoting will be confirmed by the experiments (see Section 6).

The computational cost of `solvemat` is of order $O(N^3)$ (as it is shown in Section 5), the same asymptotical cost of the Gauss method applied directly to solve the two systems (4). But it can be greatly reduced by exploiting the Toeplitz-like structure of A.

3 Toeplitz-like matrices

The definition of Toeplitz-like structure is based on the concept of displacement rank, which measures how close a matrix is to a Toeplitz matrix. Let A be an $n \times n$ matrix and let Z be the $n \times n$ *down-shift* matrix

$$Z = \begin{bmatrix} 0 & & & \\ 1 & 0 & & \\ & \ddots & \ddots & \\ & & 1 & 0 \end{bmatrix},$$

which satisfies

$$Z^T Z = I_n - \boldsymbol{e}_n \boldsymbol{e}_n{}^T \quad \text{and} \quad Z Z^T = I_n - \boldsymbol{e}_1 \boldsymbol{e}_1{}^T, \tag{6}$$

where $\boldsymbol{e}_i$ is the ith canonical vector of size n. We consider the *displacement operator*

$$\nabla(A) = A - ZAZ^T, \tag{7}$$

and the *displacement rank* $r(A) = \operatorname{rank} \nabla(A)$. Let $\rho = r(A)$, then

$$\nabla(A) = C\,D^T, \tag{8}$$

for suitable $n \times \rho$ matrices C and D. Denoting by $\boldsymbol{c}_i$ and $\boldsymbol{d}_i$ the columns of C and D respectively, then

$$\nabla(A) = \sum_{i=1}^{\rho} \boldsymbol{c}_i \boldsymbol{d}_i^T.$$

For example, let A be the Toeplitz matrix whose elements are $a_{i,j} = a_{j-1}$, with $i, j = 1, \ldots, n$. In this case

$$\nabla(A) = \boldsymbol{e}_1\,\boldsymbol{d}_1^T + \boldsymbol{c}_2\,\boldsymbol{e}_1^T, \quad \text{with} \quad \boldsymbol{d}_1 = ZZ^T A^T \boldsymbol{e}_1, \quad \boldsymbol{c}_2 = A\,\boldsymbol{e}_1.$$

$\boldsymbol{d}_1^T$ is the first row of A with the first component replaced by 0, and $\boldsymbol{c}_2$ is the first column of A. Hence, the displacement rank of a Toeplitz matrix is $\rho = 2$, except in the special case of a lower or an upper triangular matrix, where $\rho = 1$.

The matrices C and D are called *generators* of A and the matrix A is said to be *Toeplitz-like* if ρ is small relative to the size n (more formally $\rho = O(1)$ for $n \to \infty$). The set of Toeplitz-like matrices, unlike the set of Toeplitz matrices, is closed under the operations of multiplication and inversion.

The generators enable us to represent a Toeplitz-like matrix as the sum of products of lower and upper triangular Toeplitz factors. Denoting by $L(\boldsymbol{s})$ the lower triangular Toeplitz matrix whose first column is $\boldsymbol{s}$ and by $L^T(\boldsymbol{s})$ the upper triangular Toeplitz matrix whose first row is $\boldsymbol{s}$, then

$$A = \sum_{i=1}^{\rho} A_i, \quad \text{where} \quad A_i = L(\boldsymbol{c}_i)\,L^T(\boldsymbol{d}_i). \tag{9}$$

Relation (9) allows us to compute the product of a Toeplitz-like matrix by a vector, multiplying first upper and then lower triangular Toeplitz matrices by vectors. Exploiting the properties of the circulant matrices, these products can be performed at low cost using FFT. Since from (7) and (8)

$$\nabla(A^T) = \nabla(A)^T = D\,C^T, \tag{10}$$

we can compute the product $A^T\boldsymbol{v}$ by just swapping the generators of A.

In the following we assume that the elements of A are not given explicitly but only through the generators C, D. If some element of A is required, it must be computed. For example, the jth column of A is given by

$$A\boldsymbol{e}_j = \sum_{i=1}^{\rho} L(\boldsymbol{c}_i)\,\boldsymbol{v}_j^{(i)}, \text{ where } \boldsymbol{v}_j^{(i)} = L^T(\boldsymbol{d}_i)\,\boldsymbol{e}_j = \left[\, d_{j,i},\, \ldots,\, d_{1,i},\, \boldsymbol{0}_{n-j}^T \right]^T. \tag{11}$$

The representation of a matrix by means of the generators is not unique. For example, more than ρ vectors might be used to represent a matrix having a displacement rank ρ. This frequently happens when we add or multiply Toeplitz-like matrices without performing a previous analysis of the effective rank of the result. It is important to have at any phase of the algorithm a *minimal* representation, i.e. by means of the minimum number of generators. For this reason we find here the minimal representation of the matrices involved in algorithm `solvemat`.

First we find the relations between the generators of the matrix A whose displacement rank is ρ and of its submatrices A_{11}, A_{12}, A_{21}, A_{22} as indicated in (2). Let the generators C and D of A be compatibly partitioned

$$C = \begin{bmatrix} \overline{C} \\ \underline{C} \end{bmatrix}, \qquad D = \begin{bmatrix} \overline{D} \\ \underline{D} \end{bmatrix}.$$

For simplicity sake, we still denote by Z the down-shift matrix of order $n/2$ and by $\boldsymbol{e}_i$ the ith canonical vector of length $n/2$.

- The displacements of the blocks are

$$\begin{aligned} &\nabla(A_{11}) = \overline{C}\,\overline{D}^{\,T}, \quad \nabla(A_{12}) = \overline{C}\,\underline{D}^T + \boldsymbol{v}_1\boldsymbol{e}_1^T, \\ &\nabla(A_{21}) = \underline{C}\,\overline{D}^{\,T} + \boldsymbol{e}_1\boldsymbol{v}_2^T, \quad \nabla(A_{22}) = \underline{C}\,\underline{D}^T + \boldsymbol{v}_3\boldsymbol{e}_1^T + \boldsymbol{e}_1\boldsymbol{v}_4^T, \end{aligned} \tag{12}$$

where

$$\begin{aligned} &\boldsymbol{v}_1 = ZA_{11}\boldsymbol{e}_{n/2}, \quad \boldsymbol{v}_2 = ZA_{11}^T\boldsymbol{e}_{n/2}, \quad \boldsymbol{v}_3 = \alpha\,\boldsymbol{e}_1 + ZA_{21}\boldsymbol{e}_{n/2}, \\ &\boldsymbol{v}_4 = ZA_{12}^T\boldsymbol{e}_{n/2}, \quad \alpha = \boldsymbol{e}_{n/2}^T A_{11}\boldsymbol{e}_{n/2}. \end{aligned} \tag{13}$$

The relations follow directly from the partition

$$\nabla(A) = A - ZAZ^T = \left[\begin{array}{c|c} A_{11} - ZA_{11}Z^T & A_{12} - ZA_{12}Z^T - \boldsymbol{v}_1\boldsymbol{e}_1^T \\ \hline A_{21} - ZA_{21}Z^T - \boldsymbol{e}_1\boldsymbol{v}_2^T & A_{22} - ZA_{22}Z^T - \boldsymbol{v}_3\boldsymbol{e}_1^T - \boldsymbol{e}_1\boldsymbol{v}_4^T \end{array}\right].$$

Hence, the displacement ranks of the blocks are

$$r(A_{11}) = \rho, \quad r(A_{12}) = r(A_{21}) = \rho + 1, \quad r(A_{22}) = \rho + 2,$$

the generators are

$$\begin{aligned} &C_{11} = \overline{C}, \quad D_{11} = \overline{D}, && C_{12} = \big[\overline{C}\,, \boldsymbol{v}_1\big], \quad D_{12} = \big[\underline{D}\,, \boldsymbol{e}_1\big], \\ &C_{21} = \big[\underline{C}\,, \boldsymbol{e}_1\big],\ D_{21} = \big[\overline{D}\,, \boldsymbol{v}_2\big], && C_{22} = \big[\underline{C}\,, \boldsymbol{v}_3\,, \boldsymbol{e}_1\big],\ D_{22} = \big[\underline{D}\,, \boldsymbol{e}_1, \boldsymbol{v}_4\big]. \end{aligned} \tag{14}$$

- The displacement of $F = A_{11}^{-1}A_{12}$ is

$$\begin{aligned} \nabla(F) = F - ZFZ^T &= A_{11}^{-1}\big(A_{12} - A_{11}ZFZ^T\big) \\ &= A_{11}^{-1}\big(\nabla(A_{12}) + ZA_{11}FZ^T - A_{11}ZFZ^T\big) \end{aligned}$$

By using (6) we get

$$\begin{aligned}\nabla(F) &= A_{11}^{-1}\big(\nabla(A_{12}) + ZA_{11}(Z^TZ + \boldsymbol{e}_{n/2}\boldsymbol{e}_{n/2}^T)FZ^T - A_{11}ZFZ^T\big)\\ &= A_{11}^{-1}\big(\nabla(A_{12}) - \nabla(A_{11})ZFZ^T + \boldsymbol{v}_1\boldsymbol{e}_{n/2}^TFZ^T\big),\end{aligned}$$

and applying (14) we get

$$\nabla(F) = A_{11}^{-1}\overline{C}\left(\underline{D} - ZF^TZ^T\overline{D}\right)^T + A_{11}^{-1}\boldsymbol{v}_1\boldsymbol{f}^T, \quad \text{where} \quad \boldsymbol{f} = \boldsymbol{e}_1 + ZF^T\boldsymbol{e}_{n/2},$$

Hence the displacement rank of F is in general $r(F) = \rho + 1$ and its generators are

$$\begin{aligned}&C_F = A_{11}^{-1}C_{12},\\ &D_F = \left[\underline{D} - ZA_{12}^TA_{11}^{-T}Z^T\overline{D}\,, \boldsymbol{f}\right] = D_{12} - ZA_{12}^TA_{11}^{-T}\left[Z^T\overline{D}\,, -\boldsymbol{e}_{n/2}\right].\end{aligned} \tag{15}$$

- The displacement of $G = A_{21}A_{11}^{-1}$ is

$$\begin{aligned}\nabla(G) &= G - ZGZ^T = G(ZZ^T + \boldsymbol{e}_1\boldsymbol{e}_1^T) - ZA_{21}A_{11}^{-1}Z^T\\ &= GZA_{11}Z^TZA_{11}^{-1}Z^T + GZA_{11}\boldsymbol{e}_{n/2}\boldsymbol{e}_{n/2}^TA_{11}^{-1}Z^T + G\boldsymbol{e}_1\boldsymbol{e}_1^T\\ &\quad -ZA_{21}(Z^TZ + \boldsymbol{e}_{n/2}\boldsymbol{e}_{n/2}^T)A_{11}^{-1}Z^T\\ &= \big(GZA_{11}Z^T - ZA_{21}Z^T\big)ZA_{11}^{-1}Z^T + G\boldsymbol{e}_1\boldsymbol{e}_1^T\\ &\quad +\big(GZA_{11}\boldsymbol{e}_{n/2} - ZA_{21}\boldsymbol{e}_{n/2}\big)\boldsymbol{e}_{n/2}^TA_{11}^{-1}Z^T.\end{aligned}$$

Setting $\boldsymbol{g} = \boldsymbol{v}_3 - G\boldsymbol{v}_1$, $\boldsymbol{h} = G\boldsymbol{e}_1$ and $\boldsymbol{k} = ZA_{11}^{-T}\boldsymbol{e}_{n/2}$, and applying (7) and (13) we get

$$\nabla(G) = \big[\nabla(A_{21}) - G\nabla(A_{11})\big]ZA_{11}^{-1}Z^T + \big(\alpha\boldsymbol{e}_1 - \boldsymbol{g}\big)\,\boldsymbol{k}^T + \boldsymbol{h}\,\boldsymbol{e}_1^T.$$

Now we note that the first columns of $\nabla(A_{21})$ and of $\nabla(A_{11})$ coincide with the first columns of A_{21} and of A_{11} respectively. Then

$$\big[\nabla(A_{21}) - G\nabla(A_{11})\big]\boldsymbol{e}_1 = A_{21}\boldsymbol{e}_1 - A_{21}A_{11}^{-1}A_{11}\boldsymbol{e}_1 = 0,$$

i.e. by (12)

$$\big(\underline{C} - G\overline{C}\big)\overline{D}^{\,T}\boldsymbol{e}_1 + \boldsymbol{e}_1\boldsymbol{v}_2^T\boldsymbol{e}_1 = 0.$$

Since $\boldsymbol{v}_2^T\boldsymbol{e}_1 = 0$, the $n \times \rho$ matrix $\underline{C} - G\overline{C}$ has rank $\rho - 1$. Let $\boldsymbol{s} = \overline{D}^T\boldsymbol{e}_1$ be the first row of $\overline{D}$. If $\boldsymbol{s}$ has an element $s_j = 0$, the jth column of $\underline{C} - G\overline{C}$ can be discarded. Otherwise, for an index $j \in \{1, \ldots, \rho\}$, let $\widehat{\boldsymbol{s}}$ be the vector obtained by dropping the jth component of the vector whose elements are $s_i/\overline{d}_{1,j}$, $i = 1, \ldots, \rho$. Then

$$\underline{C} - G\overline{C} = \big(\underline{C} - G\overline{C}\big)PQ^T,$$

where P is the $\rho \times (\rho - 1)$ matrix obtained by dropping the jth column of I_ρ and Q is the $\rho \times (\rho - 1)$ obtained by replacing the jth row of P with the vector $-\widehat{s}^T$.

Any nonzero element $\overline{d}_{1,j}$ of the first row of $\overline{D}$ can be chosen as a pivot for the matrix Q. But, for stability reasons, it is preferable to choose an index j which avoids a too large increase of the right generators. For simplicity of notation, we assume that $j = \rho$, in this case

$$P = \begin{bmatrix} I_{\rho-1} \\ \mathbf{0}^T \end{bmatrix}, \qquad Q = \begin{bmatrix} I_{\rho-1} \\ -\widehat{\boldsymbol{s}}^T \end{bmatrix} = P - \widehat{\boldsymbol{e}}_\rho\, \widehat{\boldsymbol{s}}^T, \tag{16}$$

$\widehat{\boldsymbol{e}}_\rho$ being the ρth canonical vector of length ρ.

From (16) and (7) it follows

$$\begin{aligned}\big(\nabla(A_{21}) - G\nabla(A_{11})\big)ZA_{11}^{-1}Z^T &= \big(\underline{C} - G\overline{C}\big)\overline{D}^{\,T}ZA_{11}^{-1}Z^T + \boldsymbol{e}_1\boldsymbol{v}_2^TZA_{11}^{-1}Z^T \\ &= \big(\underline{C} - G\overline{C}\big)PQ^T\,\overline{D}^{\,T}ZA_{11}^{-1}Z^T - \alpha\,\boldsymbol{e}_1\,\boldsymbol{e}_{n/2}^TA_{11}^{-1}Z^T\end{aligned}$$

and

$$\nabla(G) = \big(\underline{C} - G\overline{C}\big)P\left(ZA_{11}^{-T}Z^T\,\overline{D}\,Q\right)^T - \boldsymbol{g}\,\boldsymbol{k}^T + \boldsymbol{h}\,\boldsymbol{e}_1^T.$$

Hence the displacement rank of G is in general $r(G) = \rho + 1$ and its generators are

$$C_G = \big[\big(\underline{C} - A_{21}A_{11}^{-1}\overline{C}\big)P\,,\boldsymbol{g}\,,\boldsymbol{h}\big], \quad D_G = \big[ZA_{11}^{-T}Z^T\,\overline{D}Q\,,-\boldsymbol{k}\,,\boldsymbol{e}_1\big]. \tag{17}$$

• Let $S = A_{22} - A_{21}A_{11}^{-1}A_{12}$ be the Schur complement of A_{11} in A. The displacement of S is

$$\begin{aligned}&\nabla(S) = \nabla(A_{22}) - A_{21}F + ZA_{21}FZ^T \\ &= \nabla(A_{22}) - A_{21}\nabla(F) - \nabla(A_{21})ZFZ^T + ZA_{21}\boldsymbol{e}_{n/2}\boldsymbol{e}_{n/2}^TFZ^T \\ &= \underline{C}\,\underline{D}^T + \boldsymbol{v}_3\boldsymbol{e}_1^T + \boldsymbol{e}_1\boldsymbol{v}_4^T - G\overline{C}\left(\underline{D}^T - \overline{D}^{\,T}ZFZ^T\right) - A_{21}A_{11}^{-1}\boldsymbol{v}_1\boldsymbol{f}^T \\ &\quad -\big(\underline{C}\,\overline{D}^{\,T} + \boldsymbol{e}_1\boldsymbol{v}_2^T\big)ZFZ^T + \big(\boldsymbol{v}_3 - \alpha\boldsymbol{e}_1\big)\boldsymbol{e}_{n/2}^TFZ^T \\ &= \big(\underline{C} - G\overline{C}\big)\left(\underline{D}^T - \overline{D}^{\,T}ZFZ^T\right) + \big(\boldsymbol{v}_3 - G\boldsymbol{v}_1\big)\boldsymbol{f}^T + \boldsymbol{e}_1\boldsymbol{t}^T,\end{aligned}$$

where $\boldsymbol{t} = \boldsymbol{v}_4 - ZF^TZ^T\boldsymbol{v}_2 - \alpha ZF^T\boldsymbol{e}_{n/2} = \mathbf{0}$. Using the matrices P and Q defined in (16) to reduce the representation of the first term, we have

$$\nabla(S) = \big(\underline{C} - G\overline{C}\big)\,P\left((\underline{D} - ZF^TZ^T\overline{D})\,Q\right)^T + \boldsymbol{g}\,\boldsymbol{f}^T.$$

Hence, the displacement rank of S is in general $r(S) = \rho$ and its generators are

$$C_S = \big[\big(\underline{C} - A_{21}A_{11}^{-1}\overline{C}\big)P\,,\boldsymbol{g}\big], \quad D_S = \big[\big(\underline{D} - ZA_{12}^TA_{11}^{-T}Z^T\,\overline{D}\big)Q\,,\boldsymbol{f}\big]. \tag{18}$$

4 The recursive algorithm using the generators

From (15), (17) and (18) we see that the computation of the generators of F, G and S requires solving two systems, one with matrix A_{11} and one with matrix A_{11}^T. From (10) it follows that the two matrices A_{11} and A_{11}^T have the same generators, just swapped. Hence it is only the order by which the generators are used that specify if the system we are solving has a matrix or its transpose.

The function `solvegen` reformulates the function `solvemat` of Section 2 for a matrix $A^{(k)}$ whose generators are $C^{(k)}$, $D^{(k)}$, replacing the operations on explicit matrices with the corresponding operations on the generators. The following functions are required:

- `elemgen` to solve the elementary systems with matrices of size n_e.
- `drop` to drop the indicated column from a matrix.
- `prod` to multiply a Toeplitz-like matrix by vectors.

`function` $\left[X^{(k)}, X_t^{(k)}\right] =$ `solvegen` $\left(k, C^{(k)}, D^{(k)}, B^{(k)}, B_t^{(k)}\right)$

`if` $k = p - q$

$\left[X^{(k)}, X_t^{(k)}\right] =$ `elemgen` $\left(C^{(k)}, D^{(k)}, B^{(k)}, B_t^{(k)}\right)$;

`else`

$m = 2^{p-k}$; $\quad r_1 = (1 : m/2)$; $\quad r_2 = (m/2 + 1 : n)$;

$\overline{C} = C^{(k)}(r_1, :)$; $\ \underline{C} = C^{(k)}(r_2, :)$; $\ \overline{D} = D^{(k)}(r_1, :)$; $\ \underline{D} = D^{(k)}(r_2, :)$;

compute the vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ defined in (13),

$C_{11} = \overline{C}$; $\quad D_{11} = \overline{D}$; $\quad C_{12} = \left[\overline{C}\,, \boldsymbol{v}_1\right]$; $\quad D_{12} = \left[\underline{D}\,, \boldsymbol{e}_1\right]$;

$C_{21} = \left[\underline{C}\,, \boldsymbol{e}_1\right]$; $\ D_{21} = \left[\overline{D}\,, \boldsymbol{v}_2\right]$;

compute the vector $\boldsymbol{v}_3$ defined in (13), $\quad C_+ = \left[\underline{C}\,, \boldsymbol{v}_3\right]$;

$\overline{B} = B^{(k)}(r_1, :)$; $\ \overline{B}_t = B_t^{(k)}(r_1, :)$; $\ \underline{B} = B^{(k)}(r_2, :)$; $\ \underline{B}_t = B_t^{(k)}(r_2, :)$;

$B = \left[\,\overline{B}\,, C_{12}\,, \boldsymbol{e}_1\right]$; $\quad B_t = \left[\,\overline{B}_t\,, Z^T\,\overline{D}\,, \boldsymbol{e}_{n/2}\right]$;

$\left[W\,, W_t\right] =$ `solvegen`$\left(k+1, C_{11}, D_{11}, B, B_t\right)$;

$[\,U, C_F, \boldsymbol{u}\,] = W$; $\quad [\,U_t, D_t, \boldsymbol{u}_t\,] = W_t$;

$Y = \underline{B} -$ `prod` $\left(C_{21}, D_{21}, U\right)$; $\quad Y_t = \underline{B}_t -$ `prod` $\left(D_{12}, C_{12}, U_t\right)$;

$\widehat{\boldsymbol{s}} = \overline{D}(1, 1 : \rho - 1)/\overline{D}(1, \rho)$; $\quad \boldsymbol{k} = Z\,\boldsymbol{u}_t$;

$D_G = \left[Z\left(\texttt{drop}\left(D_t, \rho\right) - D_t(\,:, \rho)\,\widehat{\boldsymbol{s}}^T\right), -\boldsymbol{k}\,, \boldsymbol{e}_1\right]$;

$D_u = \underline{D} - Z$ `prod` $\left(D_{12}, C_{12}, D_t\right)$;

$\boldsymbol{f} = \boldsymbol{e}_1 + Z$ `prod` $\left(D_{12}, C_{12}, \boldsymbol{u}_t\,\right)$;

$D_F = \left[D_u\,, \boldsymbol{f}\right]$;

$C_S = C_+ -$ `prod` $\left(C_{21}, D_{21}, C_F\right)$; $\quad \boldsymbol{h} =$ `prod` $\left(C_{21}, D_{21}, \boldsymbol{u}\right)$;

$C_S = \texttt{drop}\,(C_S, \rho)$; $\quad C_G = [C_S\,, \boldsymbol{h}]$;

$D_S = \left[\,\texttt{drop}\,(D_u, \rho) - D_u(:, \rho)\,\widehat{\boldsymbol{s}}^{\,T}\,, \boldsymbol{f}\right]$;

$\left[\underline{X},\, \underline{X}_{\,t}\right] \;=\; \texttt{solvegen}\,(k+1,\, C_S,\, D_S,\, Y,\, Y_t)$;

$\overline{X} = U - \texttt{prod}\,(C_F\,,\, D_F\,,\, \underline{X})$; $\quad \overline{X}_t = U_t - \texttt{prod}\,(D_G,\, C_G\,,\underline{X}_{\,t})$;

$X^{(k)} = \begin{bmatrix} \overline{X} \\ \underline{X} \end{bmatrix}$; $\quad X_t^{(k)} = \begin{bmatrix} \overline{X}_t \\ \underline{X}_{\,t} \end{bmatrix}$;

`end`

The first time, the function is applied with $k = 0$, $C^{(0)} = C$, $D^{(0)} = D$, $B^{(0)} = \boldsymbol{b}$ and a void vector $B_t^{(0)}$. At the elementary level, any routine can be chosen as `elemgen` to solve the systems having coefficient matrix of size 2^q. A natural choice would be Levinson algorithm, which can be applied directly to the generators. If stability is at risk, a more stable method, like Gaussian reduction, can be chosen, but in this case the reconstruction of the elementary matrix from the generators is required.

A rigorous error analysis of the stability of `solvegen` has not yet been performed. We entrust this matter to the numerical experimentation and examine the stability behavior of `solvegen` in comparison with that of `solvemat`.

5 Computational cost and storage requirement

The asymptotical cost for $N \to \infty$ of algorithm `solvegen` is here examined, in terms of the number of multiplicative operations. Let $\mathcal{S}_j^{(k)}$, $j = 1, \ldots, 2^k$, be the systems to be solved at the kth level of recursion, with $k = 0, \ldots, p-q$. Each system $\mathcal{S}_j^{(k)}$ has a coefficient matrix of size 2^{p-k} and $\nu_{k,j}$ right-hand sides, with $\nu_{0,1} = 1$. When $k < p - q$, from the system $\mathcal{S}_j^{(k)}$ two subsystems $\mathcal{S}_{2j-1}^{(k+1)}$ and $\mathcal{S}_{2j}^{(k+1)}$ are derived, the first one with $\nu_{k+1,2j-1} = \nu_{k,j} + \delta$ right-hand sides and the second one with $\nu_{k+1,2j} = \nu_{k,j}$ right-hand sides, where $\delta = 2\rho + 3$. The total length $\ell_k = \sum_{j=1}^{2^k} \nu_{k,j}$ of all the right-hand sides at the kth level satisfies the relation

$$\ell_{k+1} = 2\ell_k + 2^k\,\delta, \quad \text{with} \quad \ell_0 = 1,$$

implying

$$\ell_k = 2^k + k\,2^{k-1}\delta.$$

At the kth level, let $\gamma_{k,j}$ be the cost of the recursive function which solves system $\mathcal{S}_j^{(k)}$. Then

$$\gamma_{k,j} = \gamma_{k+1,2j-1} + \gamma_{k+1,2j} + \mu_{k,j}, \tag{19}$$

where $\mu_{k,j}$ is the cost of the multiplications performed at that step. These multiplications compute the products of square matrices of size 2^{p-k-1} by vectors. The number of vectors involved in these products is $2\,\nu_{k,j} + \delta + 3$ (taking into account also the computation of the vectors $\boldsymbol{v}_1$, $\boldsymbol{v}_2$ and $\boldsymbol{v}_3$).

As already said, the product of Toeplitz-like matrices by vectors can be performed using (9) and computing the product of upper and lower triangular Toeplitz matrices by vectors through FFT's. In this way the cost of computing AV, where A has size n and displacement rank ρ and V is an $n \times n_V$ matrix, with n_V depending on n, is asymptotically equal to $2\,\varphi_n\,(\rho+1)\,n_V$ where $\varphi_n \sim 4/3\,n\,\log_2 n$, is the cost of an FFT of length $2n$ (see [18]). In the present case the Toeplitz-like matrices which enter in the products have displacement rank $\rho+1$, then

$$\mu_{k,j} \sim 2\,\varphi_{2^{p-k-1}}\,(\rho+2)\,(2\,\nu_{k,j}+\delta+3).$$

The cumulative cost at the kth level is

$$\gamma_k = \sum_{j=1}^{2^k} \gamma_{k,j} = \gamma_{k+1} + \mu_k, \quad \text{where} \quad \mu_k = \sum_{j=1}^{2^k} \mu_{k,j}, \tag{20}$$

and we get

$$\begin{aligned}\mu_k &\sim 2\,\varphi_{2^{p-k-1}}(\rho+2)\sum_{j=1}^{2^k}\big(2\,\nu_{k,j}+\delta+3\big) = 2\,\varphi_{2^{p-k-1}}(\rho+2)\big(2\ell_k + 2^k(\delta+3)\big)\\ &\sim \frac{1}{3}\,2^{p+2}\delta\,(\rho+2)k(p-k).\end{aligned}$$

Denoting by $\gamma_e = \gamma_{p-q}$ the cost of solving the 2^{p-q} elementary systems, from (20) the total cost $\gamma = \gamma_0$ of the algorithm turns out to be

$$\gamma = \gamma_e + \beta, \quad \text{with} \quad \beta = \sum_{k=0}^{p-q-1} \mu_k \sim \frac{\delta\,(\rho+2)}{9}\,2^{p+1}\,p^3 \sim \frac{2\,\delta\,(\rho+2)}{9}\,N\,\log_2^3 N.$$

To evaluate γ_e we must take into consideration which algorithm is used for solving the elementary systems. If Levinson algorithm is used, we have

$$\gamma_{p-q,j} \sim 2^{2q}\,\big(\nu_{p-q,j} + \frac{5\,\rho}{2}\big),$$

$$\gamma_e^{Lev} = \sum_{j=1}^{2^{p-q}} \gamma_{p-q,j} = 2^{2q}\,\big(\ell_{p-q} + \frac{5\,\rho}{2}\,2^{p-q}\big) \sim \delta\,2^{p+q-1}\,(p-q) \sim \frac{\delta}{2}\,n_e\,N\,\log_2 N.$$

If the Gaussian reduction is used, we have

$$\gamma_e^{Gau} \sim 2^{p-q}\,\frac{2^{3q}}{3} + 2^{2q}\ell_{p-q} + \gamma_{\rm rec},$$

where $\gamma_{\rm rec}$ is the cost of the reconstruction of the elementary matrices from the generators. Using (11) and computing the products with FFT, the cost to reconstruct each elementary matrix is asymptotically equal to $\varphi_{n_e}(\rho+1)\,n_e$, implying

$$\gamma_{\rm rec} \sim 2^{p-q}\varphi_{n_e}\,(\rho+1)n_e \sim \frac{\rho+1}{3}\,2^{p+q+2}\,q.$$

Hence

$$\gamma_e^{Gau} \sim \frac{2^{p+2q}}{3} + (p-q)\,2^{p+q-1}\,\delta \sim \frac{N\,n_e^2}{3} + \frac{\delta\,n_e}{2}\,N\,\log_2 N.$$

By comparing β with γ_e^{Lev} (or γ_e^{Gau}) we can choose the size n_e of the elementary blocks in order to keep the cost of `solvegen` of the same order than β. When Levinson algorithm is used at the last level of recursion, β dominates over γ_e^{Lev} if $n_e = O(p^2)$. When Gaussian elimination is used, β dominates over γ_e^{Gau} if $n_e = O(p^{3/2})$. With these choices of n_e the cost of `solvegen` is of order $O(N \log_2^3 N)$, showing that it belongs to the class of superfast methods.

The experimentation has shown that, except in some cases especially constructed, the error which affects the computed solution decreases when the length $p-q$ of the recursion decreases, i.e. when n_e increases. Hence, from the stability point of view, a large n_e would be preferable, but cost considerations suggest to keep it small, for example by choosing n_e a constant or, at most, a multiple of $\log_2 N$. For simplicity, in the experiments we assume $q = \lceil \log_2(\rho+1) \rceil$.

The analysis of the cost of algorithm `solvemat` can be carried out in a similar way to what has been done above for algorithm `solvegen`, with the following differences:

– the increasing of the number of right-hand sides of the systems depends on the level k, i.e. $\nu_{k+1,2j-1} = \nu_{k,j} + \delta_k$, where $\delta_k = 2^{p-k}$, implying

$$\ell_k = 2^{p+k} + 2^k - 2^p, \quad \mu_k = 2^{2(p-k-1)}\big(2\ell_k + 2^{p-1}\big) \sim 2^{3p-k-3}(4 - 3\cdot 2^k),$$

and

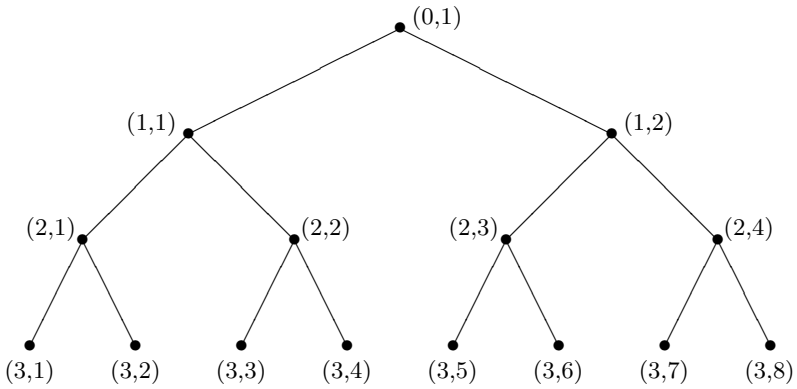
$$\beta \sim 2^{3p-1} \sim \frac{N^3}{2}.$$

– at the elementary level no reconstruction is required, then using Gauss method we have

$$\gamma_e \sim 2^{p-q}\,\frac{2^{3q}}{3} + 2^{2q}\ell_{p-q} \sim 2^{2p+q} = N^2 n_e,$$

showing that the asymptotical cost of algorithm `solvemat` is $\gamma \sim N^3/2$, of the same order of Gauss method.

To analyze the storage requirement of `solvegen`, we assume that the allocation of A is made through its generators, requiring $\lambda_0 = 2^{p+1}\rho$ memory locations. The following analysis counts:
(a) the λ_1 memory locations required by the right-hand sides and the solutions of the problems to be solved,
(b) the λ_2 memory locations required by the generators of the involved matrices.
The solution process can be described by a complete binary tree of height $p-q$ with 2^k nodes at level k. The jth node at level k is associated to the problem



$\mathcal{S}_j^{(k)}$ (i.e. a system with matrix of size 2^{p-k}), as shown in the figure. The nodes are labelled by the pair (k, j), with $j = 1, \ldots, 2^k$. So the root of the tree stands for the initial system $\mathcal{S}_1^{(0)}$ and the leaves stand for the systems $\mathcal{S}_j^{(p-q)}$ that are not divided any further but directly solved. The node (k, j) at the kth level forks into the two nodes $(k+1, 2j-1)$ and $(k+1, 2j)$ at the next level, corresponding to the two `solvegen` calls. The problem at the node (k, j) has a local requirement of $2^{p-k} \times \nu_{k,j}$ memory locations for its right-hand sides $\left[B^{(k)}, B_t^{(k)}\right]$, where $\nu_{k+1,j} \leq \nu_{k,\lceil j/2 \rceil} + \delta$, with $\delta = 2\rho + 3$ and $\nu_{0,1} = 1$. Then $\nu_{k,j} \leq (1 + k\,\delta)$.

When the problem at the node (k, j) has been solved, its solution can be overwritten on its right-hand sides and the memory locations required by its children can be released. Moreover, if j is even, also its parent problem can be solved and so on. On the contrary, if j is odd new memory must be allocated for its right brother problem. Then the memory allocated for the right-hand sides when solving the problem at the node (k, j) is at most the sum of the memory locations required by the node itself and of those required by all its ancestors plus those required by its left brother (if any), and by the left brothers (if any) of each ancestor, corresponding to pending problems. The largest amount of memory is required by one of the problems at the last level, which has $p - q$ ancestors. Then

$$\lambda_1 < 2 \sum_{k=0}^{p-q} 2^{p-k}\,(1 + k\,\delta) < 4\,N\,(1 + \delta) = 8\,N\,(\rho + 2).$$

To evaluate λ_2 we note that at a node (k, j) the generators of $A_{11}^{(k)}$ in the first recursive call can be overwritten by those of the Schur complement in the second recursive call. On the other hand, $2^{p-k+1}(\rho + 1)$ memory locations are needed to store C_F, D_F, C_G, D_G and this memory cannot be released until the problem $\mathcal{S}_j^{(k)}$ has been solved. Hence the largest amount of memory is required by one of the problems at the last level and

$$\lambda_2 < \sum_{k=1}^{p-q} 2^{p-k+1}(\rho + 1) < 2\,N\,(\rho + 1).$$

Then $\lambda_0 + \lambda_1 + \lambda_2 < 6\,N\,(2\rho + 3)$. Figure 1 shows the history of the storage allocation for a problem with $N = 2^9$, $\rho = 5$, $q = 3$. The memory effectively required is about the half of the one indicated by the bound.

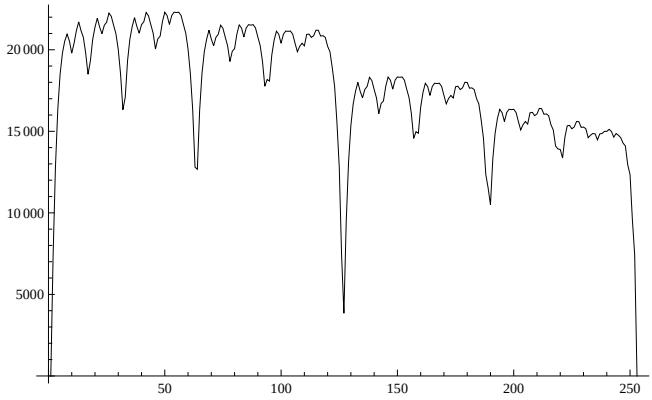


Figure 1: - History of storage allocation.

6 Numerical experiments

The experiments, which have been conducted on an Intel Core Duo @ 3 GHz, 2GB RAM, using double precision arithmetic, concern a set of 160 Toeplitz-like matrices of size 2^7, with displacement rank $\rho = 5$ and elementary size $2^q = 8$. We have performed also tests for different values of ρ, with elementary size 2^q, $q = \lceil \log_2(\rho + 1) \rceil$, but their results are not included because they do not show substantial differences. All the experiments aim at comparing the stability behavior of `solvegen` with other methods used for solving problem (1). Besides `solvegen` we have considered `solvemat` introduced in Section 2, `Gauss`, the block Gaussian elimination by columns without pivoting and block size 2^q, `Lev`, the Levinson algorithm [9, 13] and `MBA`, the Morf, Bitmead and Anderson method [2, 16]. Unlike the other methods which operate directly on the displacement representation of A, `Gauss` and `solvemat` require the reconstruction of A from the generators.

The matrices of the experiments are randomly generated through their generators C and D, in such a way to give different conditioning. This is obtained by generating $\rho - 1$ columns of C and D with uniform distribution between -1 and 1. The column $\boldsymbol{d}_\rho$ is the first canonical vector and the column $\boldsymbol{c}_\rho$ is the first canonical vector multiplied by a value λ which influences the diagonal dominance of A. The left-hand side vector $\boldsymbol{b}$ is then computed from an exact solution, randomly generated. The difficulty of any problem constructed in this way is measured through the conditioning of the elementary level blocks, by

means of the parameter

$$\kappa = \max_{j=1,\ldots,2^{p-q}} \texttt{cond}\left(A_j^{(p-q)}\right),$$

where `cond` is the condition number in ∞ norm. For the generated matrices κ varies between $10^{0.6}$ and $10^{3.6}$. The stability behavior is measured through the residual

$$R = \|\boldsymbol{b} - A\widetilde{\boldsymbol{x}}\|_\infty,$$

where $\widetilde{\boldsymbol{x}}$ is the effectively computed solution.

First we compare `solvegen` with `MBA` and `Lev`. The results shown in the Log-Log plot of Figure 2 suggest that:

- the loss of precision of the three methods increases with κ;
- even at small values of the parameter κ, `MBA` loses more precision than `solvegen` and `Lev`;
- `Lev` is usually more stable than `solvegen`, but it is asymptotically slower. Moreover, as it is pointed out in [5], `Lev` can perform poorly, regardless of the values of κ, due to the presence of ill conditioned leading principal submatrices of the original matrix. If this happens when the size of the submatrices is different from 2^{p-k}, $k = 1, \ldots, p-q$, `solvegen` outperforms `Lev`.

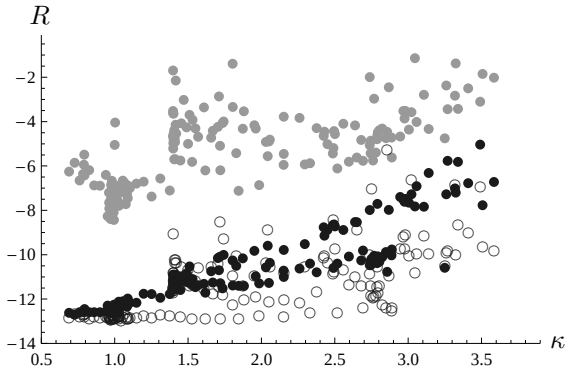


Figure 2: Log-Log plot of the residuals R as functions of κ: the black points refer to `solvegen` results, the grey points to `MBA` results, the empty circles to `Lev` results.

Then we compare `solvegen` with `solvemat` and `Gauss`. Figure 3 shows that:

- the slow increase with κ of the residuals computed by `solvemat` (grey points) closely matches the increase of the residuals computed by `Gauss` (empty circles). It turns out that `solvemat` and `Gauss` can be considered equivalent from the stability point of view.

- the worsening of the residuals of `solvegen` with respect to those of `solvemat` grows with κ.

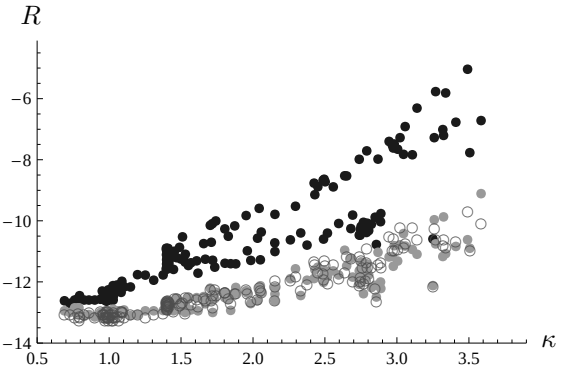


Figure 3: Log-Log plot of the residuals R as functions of κ: the black points refer to `solvegen` results, the grey points to `solvemat` results, the empty circles to `Gauss` results.

We attribute the worsening of the performance of `solvegen` to the use of the generators. It is well known that, when the operations on explicit matrices are replaced with the corresponding operations on the generators, the magnitude of the entries of the generators matrices can be related to the stability properties of the algorithm (see for example [8, 10]). The parameter

$$\sigma = \prod_{k=1}^{p-q}(1 + \max_{j=1,\ldots,2^k} \|C_j^{(k)}\|_\infty \, \|D_j^{(k)T}\|_\infty) \tag{21}$$

where $C_j^{(k)}$ and $D_j^{(k)}$ are the generators of the jth Schur complement matrix S at the kth level of recursion, turns out to be suitable in describing the stability behavior of `solvegen`. Figure 4 shows the residuals of `solvegen` (black points) and the function $\sigma\epsilon$ (grey points), ϵ being the machine precision. It appears that there is a tight relation between the behaviors of the two functions.

7 Conclusions

In this paper a divide and conquer algorithm `solvegen` has been presented for solving general Toeplitz-like systems with cost $O(N \log^3 N)$. This algorithm exploits the displacement properties of the structure and avoids the explicit inversion of the coefficient matrix. It represents an effort to achieve more stability than existing superfast algorithms, as for example the MBA algorithm [2, 16]. A theoretical error analysis has not yet been performed, but a parameter σ has been experimentally detected, which measures the magnitude of the generators of the Schur matrices involved in the computation and appears to be related to the stability of the algorithm. Further research will deepen this aspect.

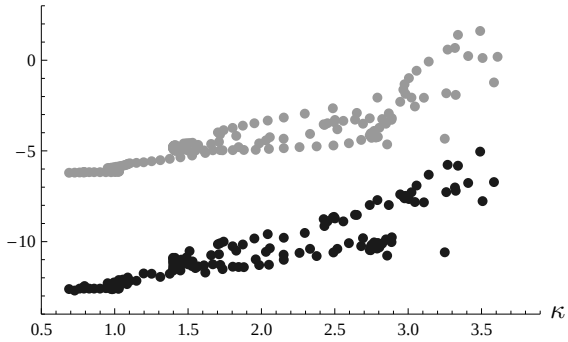


Figure 4: Log-Log plot of the residuals of `solvegen` (black points) and of $\sigma\epsilon$ (grey points) as functions of κ.

The experimentation has confirmed that `solvegen` outperforms `MBA` from the stability point of view and that it does not lose too much precision compared with Levinson algorithm, which however belongs to the class of fast algorithms. In conclusion `solvegen` appears to be a reliable superfast method when the conditioning of the elementary blocks matrices is not too large.

References

[1] G. S. Ammar and W. B. Gragg, "Superfast solution of real positive definite Toeplitz systems", *SIAM J. Matrix Anal. Appl.*, 9 (1988), pp. 6176.

[2] R.R. Bitmead and B.D.O. Anderson, "Asymptotically Fast Solution of Toeplitz and Related Systems of Linear Equations", *Linear Algebra Appl.*, 34, pp. 103-116, 1980.

[3] R. P. Brent, F. Gustavson, and D. Yun, "Fast solution of Toeplitz systems of equations and computation of Pade approximants", *J. Algorithms*, 1 (1980), pp. 259295.

[4] J.R. Bunch, "Stability of methods for solving Toeplitz systems of equations", *SIAM J. Sci. Stat. Comput.*, 6, pp. 349-364, 1985.

[5] T.F. Chan and P.C. Hansen, "A look-ahead Levinson algorithm for general Toeplitz systems", *IEEE Trans. on Signal Proc.*, 40, pp. 1079-1090, 1992.

[6] G. Cybenko, "The numerical stability of the Levinson-Durbin algorithm for Toeplitz systems of equations", *SIAM J. Sci. Stat. Comput.*, 1, pp. 303-319, 1980.

[7] F. de Hoog, A" new algorithm for solving Toeplitz systems of equations", *Linear Algebra Appl.*, 88/89 (1987), pp. 123138.

[8] P. Favati, G. Lotti and O. Menchi, "Stability of the Levinson algorithm for Toeplitz-like systems", *SIAM Journal on Matrix Analysis and Applications*, 31, pp. 2531-2552, 2010.

[9] B. Friedlander, M. Morf, T. Kailath and L. Ljung, "New Inversion formulas for matrices classified in terms of their distance from Toeplitz matrices", *Linear Algebra Appl.*, 27, pp. 31-60, 1979.

[10] M. Gu, "Stable and Efficient Algorithms for Structured Systems of Linear Equations", *SIAM Journal on Matrix Analysis and Applications*, 19, pp. 279-306, 1998.

[11] G. Heinig and K. Rost, *Algebraic methods for Toeplitz-like matrices and operators*, Akademie-Verlag, Berlin, 1984.

[12] N.J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 1996.

[13] T. Kailath, S.-Y. Kung and M. Morf, "Displacement ranks of matrices and linear equations", *J. Math. Anal. Appl.*, 68, pp. 395-407, 1979.

[14] T. Kailath and A. H. Sayed, "Displacement structure: theory and applications", *SIAM Rev.*, 37, pp. 297-386, 1995.

[15] T. Kailath, A. Viera, and M. Morf, Inverses of Toeplitz operators, innovations and orthogonal polynomials, *SIAM Rev.*, 20, pp. 106-119, 1978.

[16] M. Morf, Doubling algorithms for Toeplitz and related equations, in *Proc. IEEE Internat. conf. on Acoustics, Speech, & Signal Processing*, pp. 954-959, Denver, Colorado, 1980.

[17] B.R. Musicus, "Levinson and fast Cholesky algorithms for Toeplitz and almost Toeplitz matrices", Research Laboratory of Electronics Tec. Rep. 538, MIT, Cambridge, MA.

[18] H.V. Sorensen, D.L. Jones, M.T. Heideman, C.S. Burrus, "Real-Valued Fast Fourier Transform Algorithms", *IEEE Transactions on Acoustics, Speech, & Signal Processing* 1987; **ASSP-35**:849-863.

[19] M. Stewart, "A Superfast Toeplitz Solver with Improved Numerical Stability ", *SIAM J. Matrix Anal. Appl.*, 25, pp. 669-693, 2003.